Growth

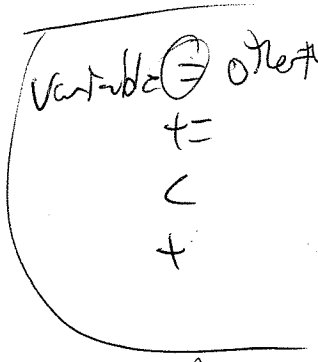
2 Resources (sp) that programs Use

1) ~~the~~ execution time

2) memory

```
int ans=0;
for (int i=0; i<n; i++){
    ans ++;
}
```

How many operations to complete function interms of n.



1 operation

3

$3N + 3$

$C_1N + C_2 \cdot 1 \Rightarrow N + 1 \Rightarrow N$

Turn into Big Oh

1. Drop constant factor
2. Drop Non-dominant terms (as N grows to ∞)

$O(N)$

$10-\sqrt{N}$, N, $N^{0.01}$, 2^N, $\log_2(N)$

$\log_e(N^2)$, $N!$

label	1	2	3	4	5	6	7
	$10-\sqrt{N}$	N	$N^{0.01}$	2^N	$\log_2(N)$	$\log_e(N^2)$	$N!$
Growth order	4	3	5	2	$\log(N)$	$\log(N^2)$	1

$\log(N)$ ~~$\log(N)$~~ 6 6

$N^{0.5}$ $N^{0.01}$

	0	1	2	3	4	5
$N!$	1	1	2	6	24	
2^N	1	2	4	8	16	

$2 < N$

$x < x$

$2x < Nx$

polynomial
$C > 0$
N^C
exponential
$C > 1$
C^N

$\log_a(c) = \frac{\log_b(c)}{\log_b(a)} = \frac{1}{\log_b(a)} \log_b(c)$ expo > poly

faster growth

N·1 $\log(N^N)$

$N \log(N)$

$\log(N)$ is not constant

it's not bounded

FTSOC k is the bound

$k < \log(10^{(k+1)}) = (k+1)\log_{10}(10)$

$= (k+1)\,1$

$= k+1$

$(\log(N))^2 \neq \log(N^N)$

$\log(N)\log(N)$ $1 \cdot \log(N)$

faster growth

BigOh formally upper bound

$f(n) = O(c_5(n))$

g(n) ~~has at least~~ grows at least as fast as f(n)

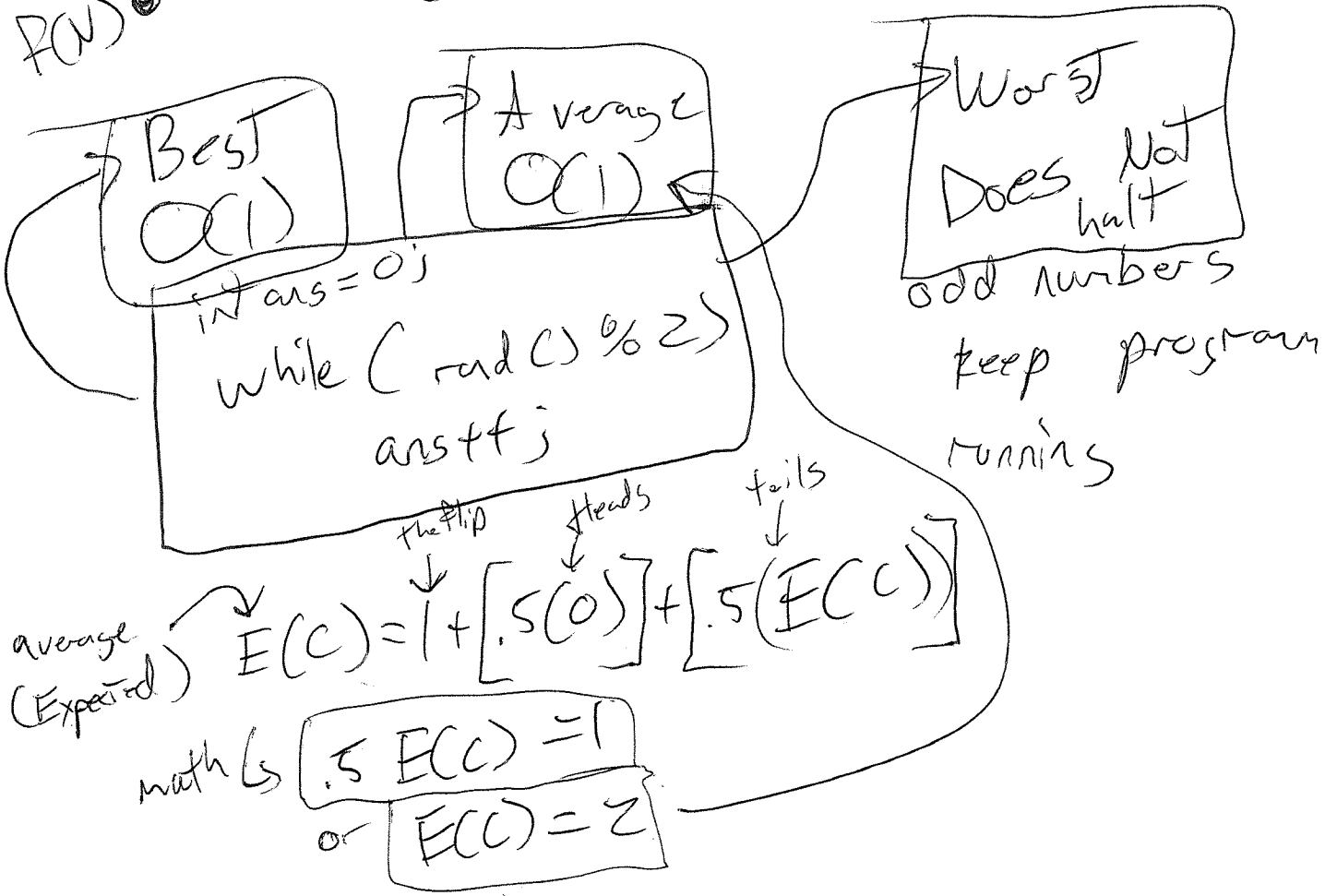
$N = O(N^2)$ ✓

$N^2 = O(N)$ X false

Big theta is exact bound

$f(N) = 2^1$
$f(N) = O(1)$

Big Omega is the lower bound



Binary Search Runtimes on N values

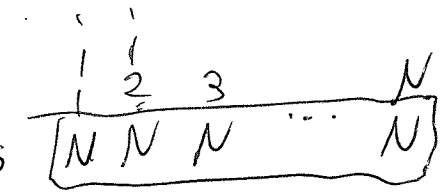
Best
$O(1)$
even if N is large
first value is target

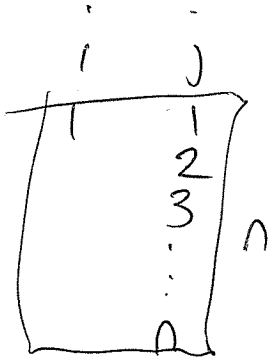
Worst
$O(\log(N))$
halving until only 1 value left

Average
$O(\log(N))$
Not obvious but
most values take $(1/2)\downarrow$ $\log(N)$ to find

```
int func1(int n) {
   int i, j, x = 0;
   for (i = 1; i <= n; i++) {
      for (j = 1; j <= n; j++) {
         x++;
      }
   }
   return x;
}
```

$O(N^2)$ ops





$N(N) = N^2 \cdot c$

```
int func2(int n) {
   int x = 0;
   for (int i = 1; i <= n; i++)
      x++;
   for (int j = 1; j <= n; j++)
      x++;
   return x;
}
```

N N $O(N)$

```
void func3(int n) {
   while (n > 0) {
      printf("%d", n % 2);
      n = n / 2;
   }
}
```

$13 \to 6 \to 3 \to 1$

1 0 1 1

1 0 4 8

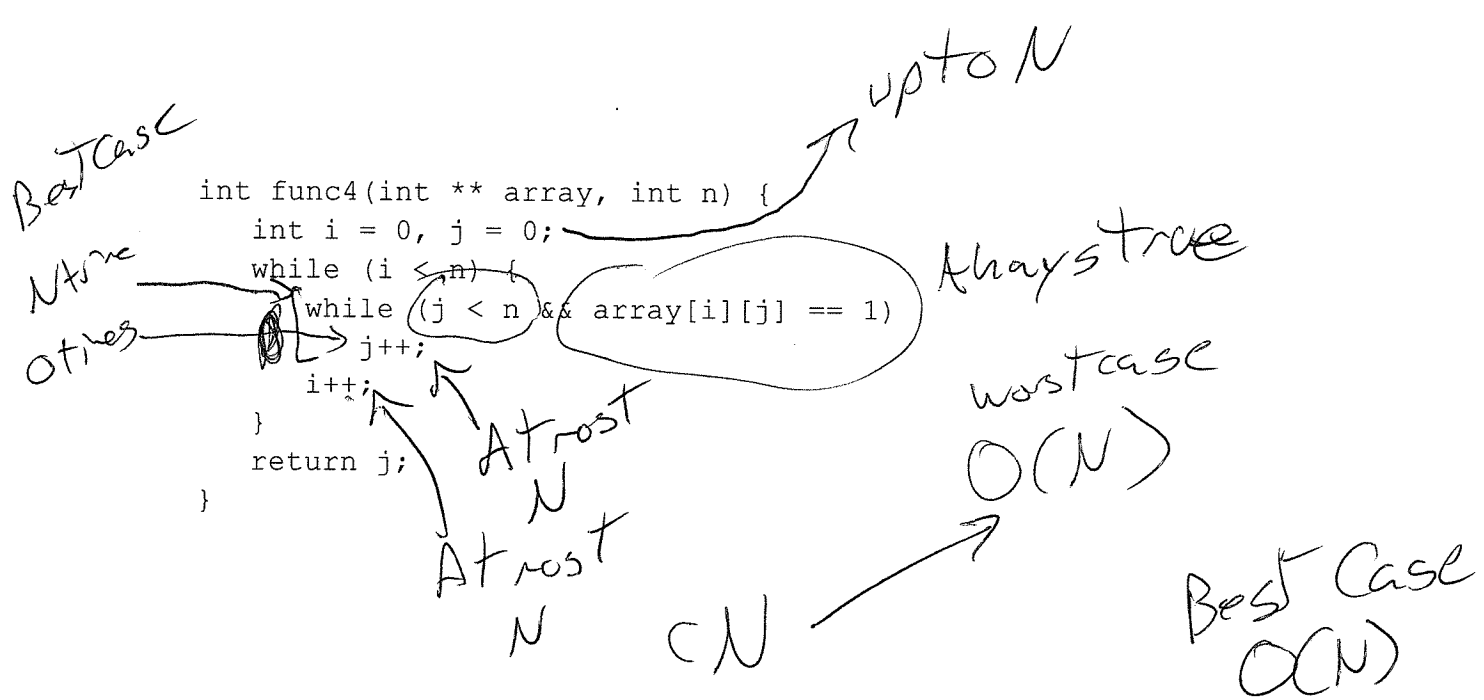
N
↳ N/2
↳ N/4
↳ N/8
...
↳ $\frac{N}{2^k}$

$\frac{N}{2^k} = 1$

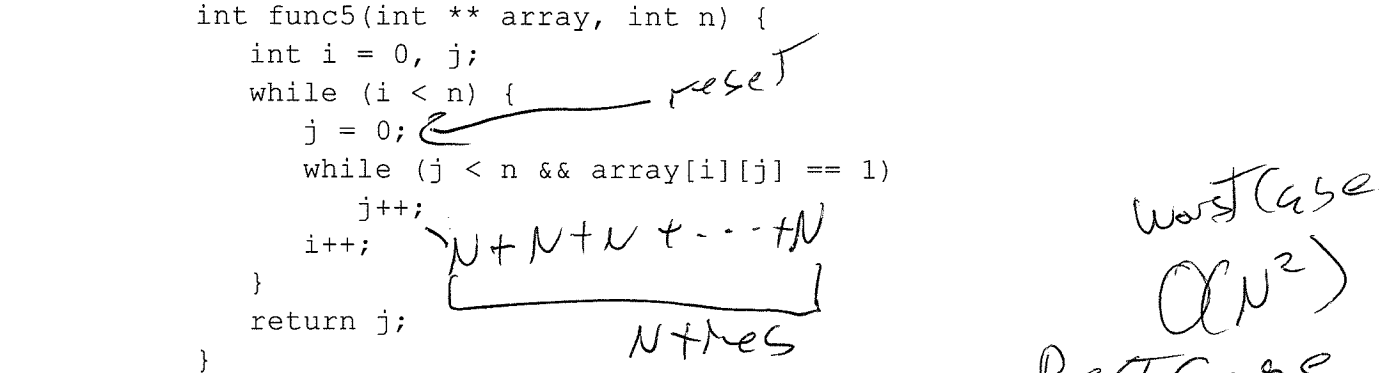
$N = 2^k$

$k = \log_2(N)$

```
int func4(int ** array, int n) {
   int i = 0, j = 0;
   while (i < n) {
      while (j < n && array[i][j] == 1)
         j++;
      i++;
   }
   return j;
}
```




```
int func5(int ** array, int n) {
   int i = 0, j;
   while (i < n) {
      j = 0;
      while (j < n && array[i][j] == 1)
         j++;
      i++;
   }
   return j;
}
```




```
int func6(int array[], int n) {
   int i, j, sum = 0;
   for (i = 0; i < n; i++)
      for (j=i+1; j < n; j++)
         if (array[i] > array[j])
            Sum++;
   return sum;
}
```